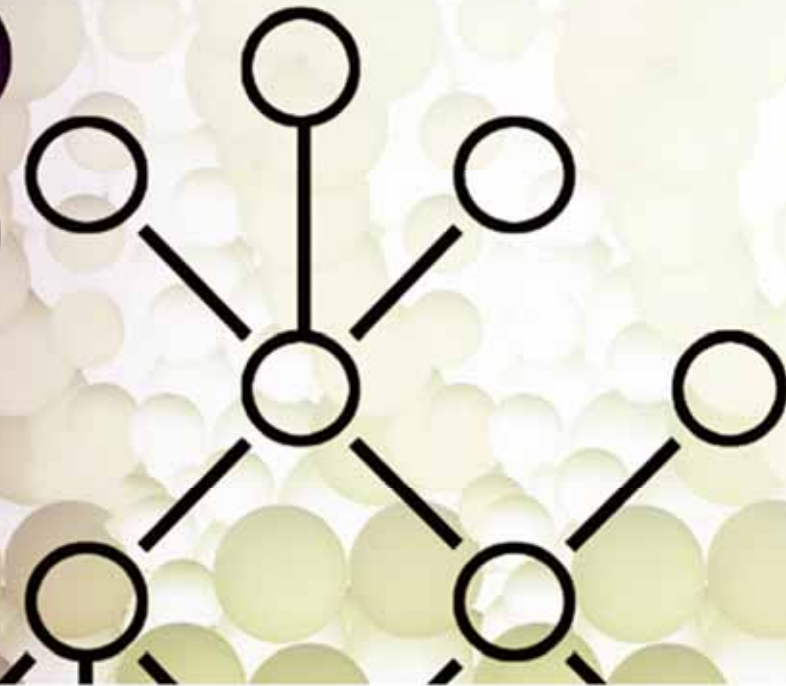




Packer Genetics: *The Selfish Code*

Ero Carrera, Jose Duarte
([ero.carrera](mailto:ero.carrera@zynamics.com), [jose.duarte](mailto:jose.duarte@zynamics.com))@zynamics.com

- Pythonized Bochs
- Compiler blood ties
- The selfish code
- Unpacking automation for fun and profit

Pythonized Bochs

Compiler blood ties

The selfish code

Unpacking automation for
fun and profit

Pythonized Bochs

- We decided to use a CPU emulator
- We chose Bochs
- Exposed the instrumentation interface to an embedded Python interpreter
- Substituted Bochs' own debugger with a Python command line

Why CPU emulation

- Hypervisor technology wasn't available when the project started
- Provides with a god-like control over the environment
- Most anti-debugging tricks are aimed at finding divergences between a real/untainted system and a monitored one
- Working at the CPU level allows us to avoid most* those (**emulators are not perfect after all*)

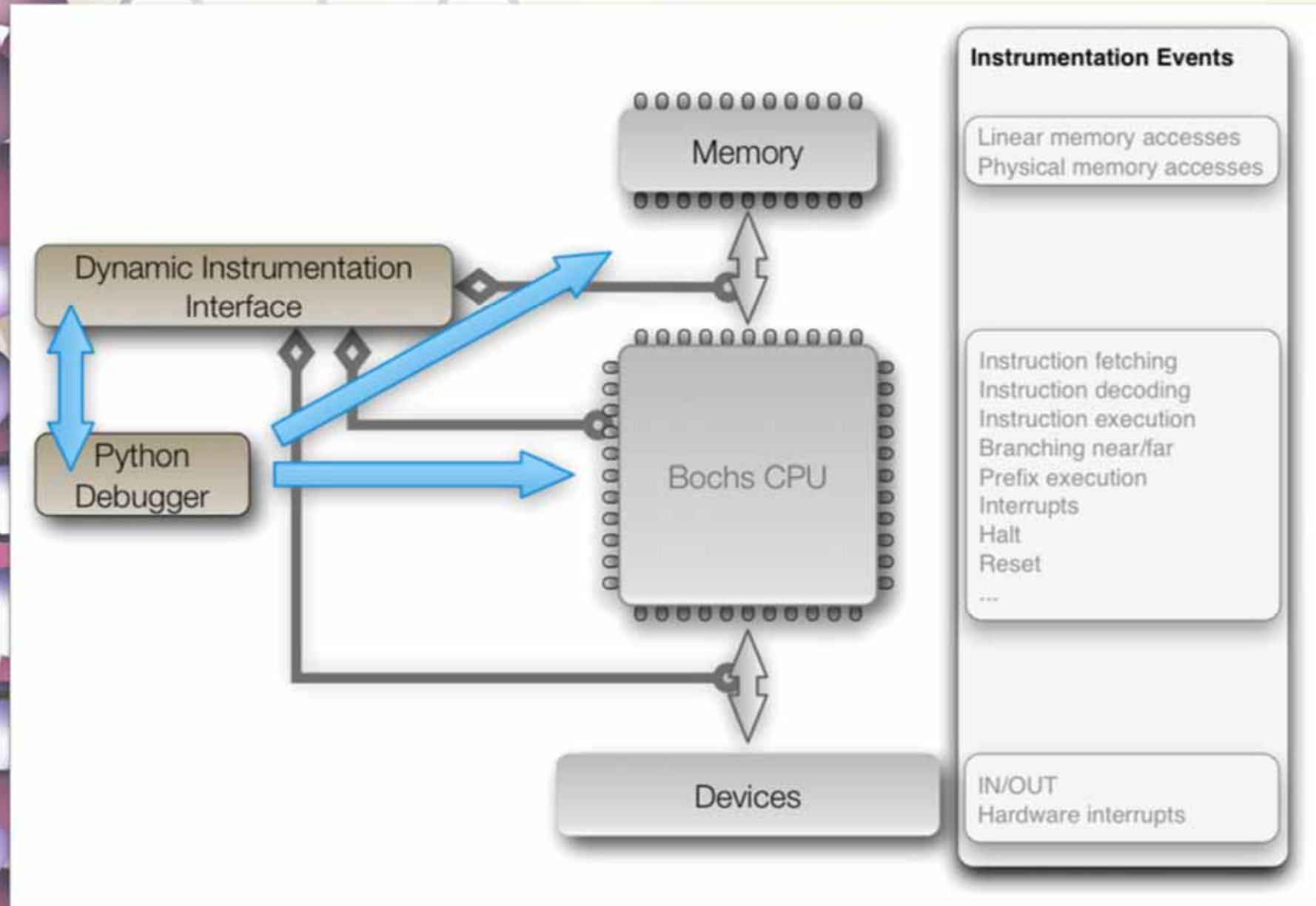
Why CPU emulation

- Sandboxes lead to the endless problems
- Need to asymptotically approach the OS feature-set to provide a veritable environment
- We didn't want to redevelop an existing OS

Bochs

- Why Bochs?
- The instrumentation interface is powerful and fine-grained
- Snapshots are neat!
- Why to embedded Python into Bochs?
- Faster development & research cycle

Overall Architecture



Drawbacks

- Speed, but entirely depends on the use case. It is not a big issue for us
- Some effort is needed in order to remove all the useless noise (other processes, kernel code, etc) as we see it all

Our approach to unpacking

- As generic as possible, the least we need to know about packers the better
- Try to algorithmically capture the generic "concept" behind unpacking
- In short: try to capture as closely as possible the:
Run -> Wait for unpacking done -> Dump
- These steps capture a vast majority of all packers if implemented with care
- We are very aware that it has limitations with packers employing runtime, on-demand unpacking/decoding techniques

Keep it simple, Stupid!

- We tried a set of different heuristics that in the end proved more problematic than helpful
- Tracking execution flow into dirty memory
 - Multi-stage, multi-layer packing a PITA
- Attempting to capture "long" jumps
 - Same problems as above and false positives
- EIP jump patterns
 - False positives

Keep it simple, Stupid!

- Monitoring specific APIs that might signal end-of-unpacking
 - Packers get to use really crazy APIs, because they need them or simply to annoy sandboxes
- Looking for typical entryptpoint patterns
 - Easy to fake and being actively done
- Therefore, those attempts that might otherwise look insightful lead to an over-specification of what we wanted to actually capture. In the end we had to go even more generic to really be universal

Issues to solve

- QoU metric (Quality of Unpacking)
- Optimize emulation time (don't keep running if it's useless)
- Filter out data/junk-code/rubbish

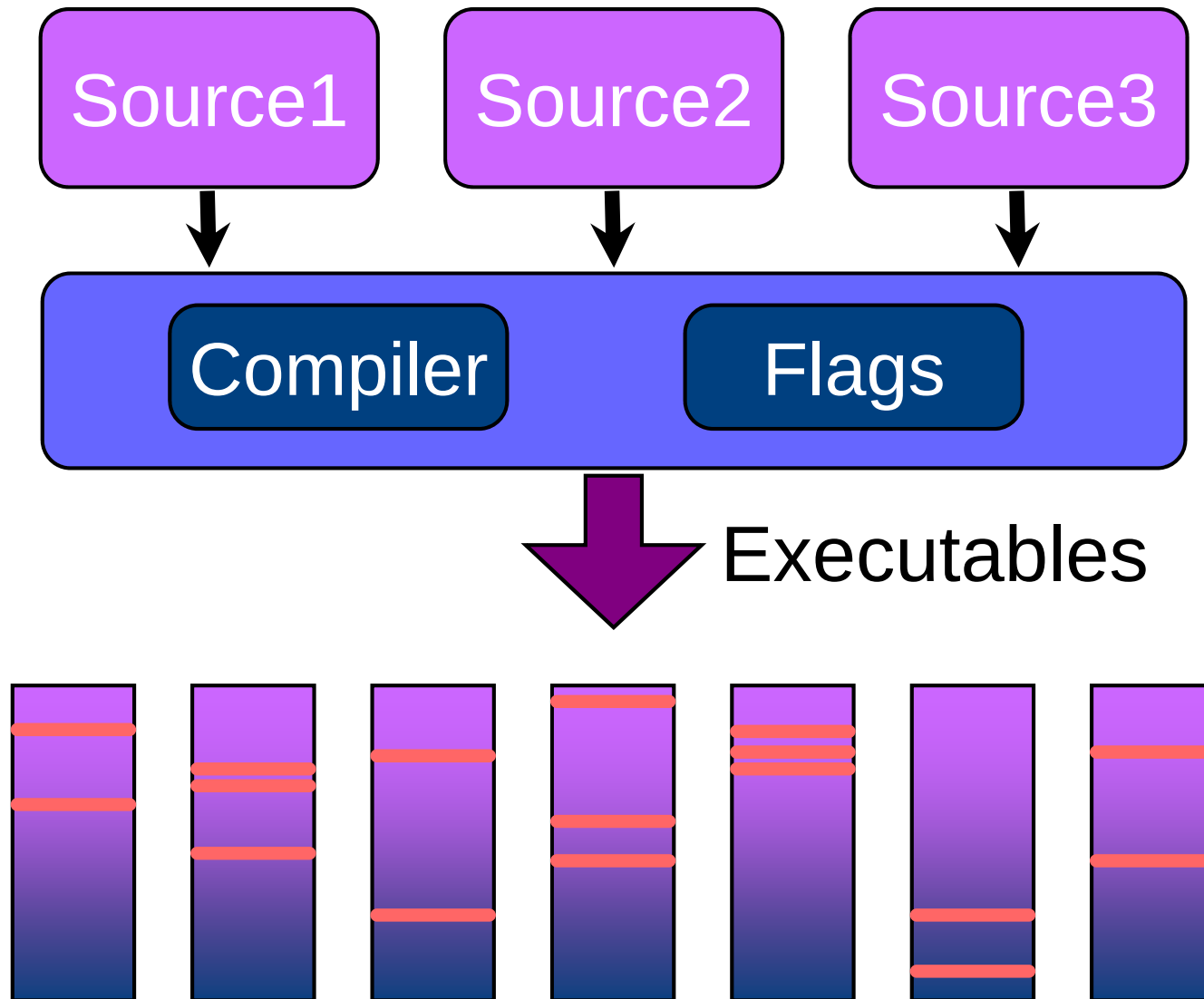
 Pythonized Bochs

 Compiler blood ties

The selfish code

Unpacking automation for
fun and profit

Compiler blood ties



Different compilers

- All compilers generate code based on functions and “imports”
- The use of functions entails:
 - Work with arguments and stack variables
 - Save and restore registers
 - Call other functions with arguments
 - ...

Different compilers

- In short... what compiler was used to generate this?

```
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
retn
```

- Hard to say, as it's common to many compilers

Same compiler

- Code-sequences surviving different compiler flags or compiler version
- Different from other compilers
- Those use to be useful to identify the exact compiler used and maybe some of the flags

```
loc_40F30D:  mov     ecx, 4  
             push    0  
             push    0  
             dec     ecx  
             jnz     short loc_40F30D
```

Blood ties

- Common patterns match common operations inside assembly-functions:
- SEH, placing arguments for callees, padding/align between functions, indirect calls (IAT-like), etc...

Gene: *“any portion of chromosomal material that potentially lasts for enough generations to serve as a unit of natural selection”*

Blood ties

- Common patterns match common operations inside assembly-functions:
- SEH, placing arguments for callees, padding/align between functions, indirect calls (IAT-like), etc...

Gene: “*any portion of binary opcodes that potentially lasts for enough binaries to serve as a unit compiler-code detection*”

 Pythonized Bochs

 Compiler blood ties

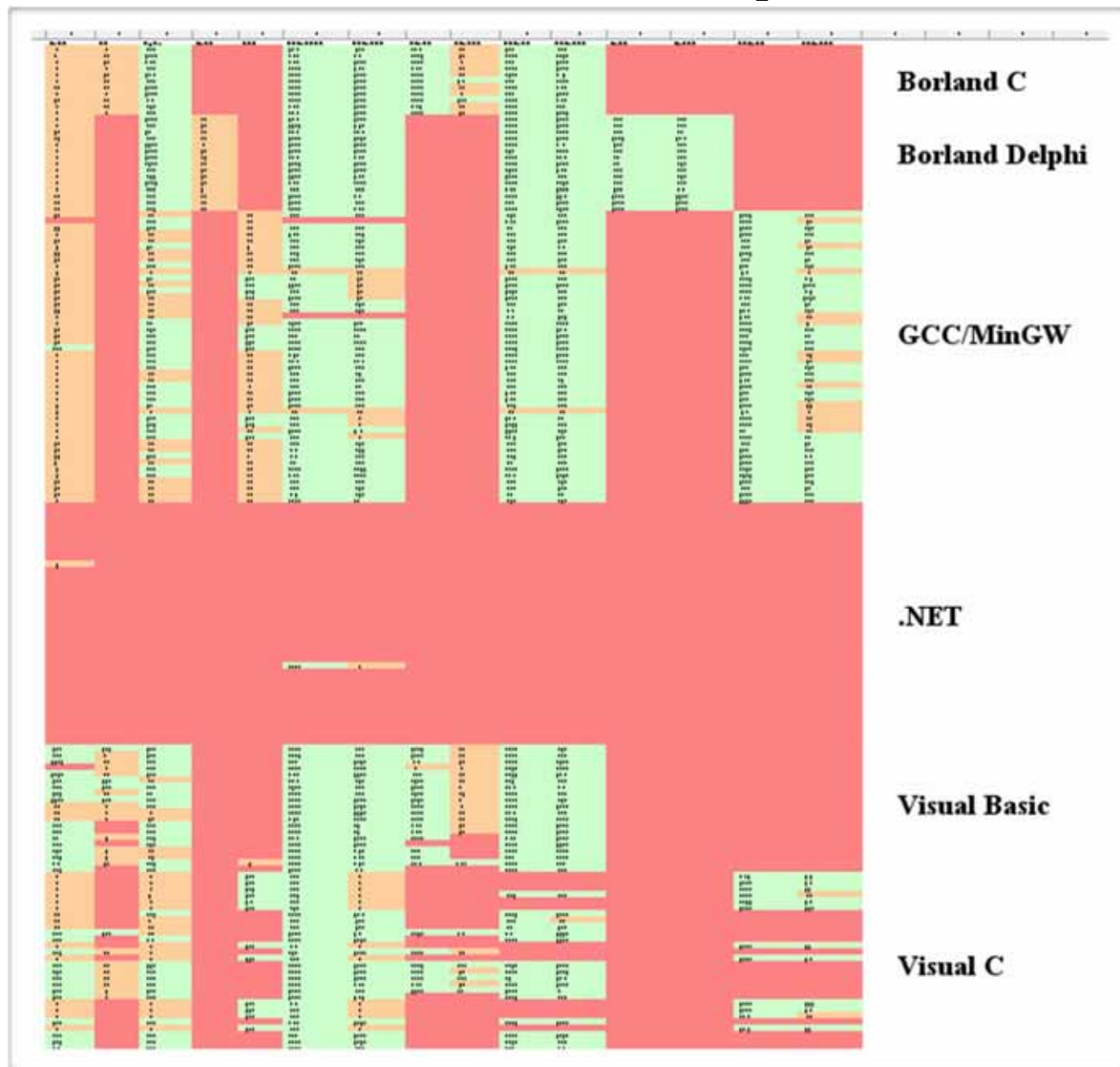
 The selfish code

Unpacking automation for
fun and profit

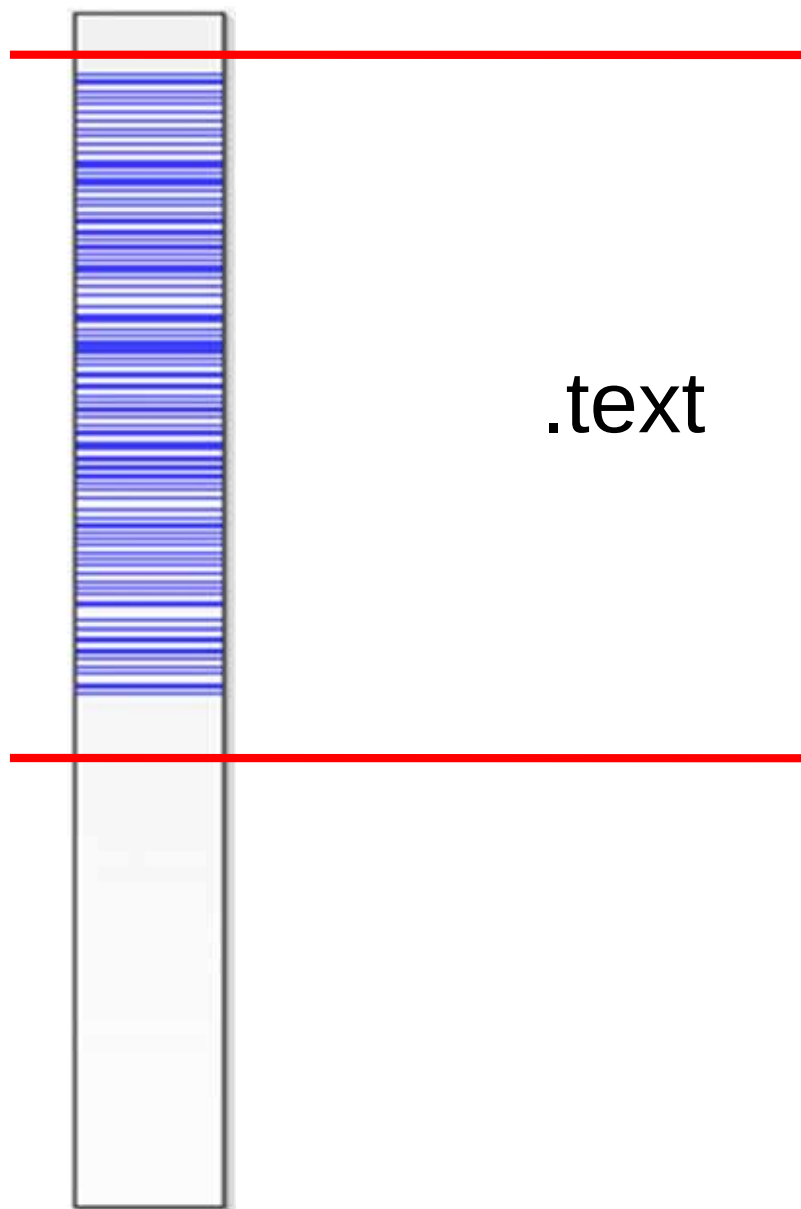
Genetics and packers

- Most packed samples contain no compiler-generated code but obfuscated-code
- Search for compiler-genes produces zero or few results

Genetics and packers

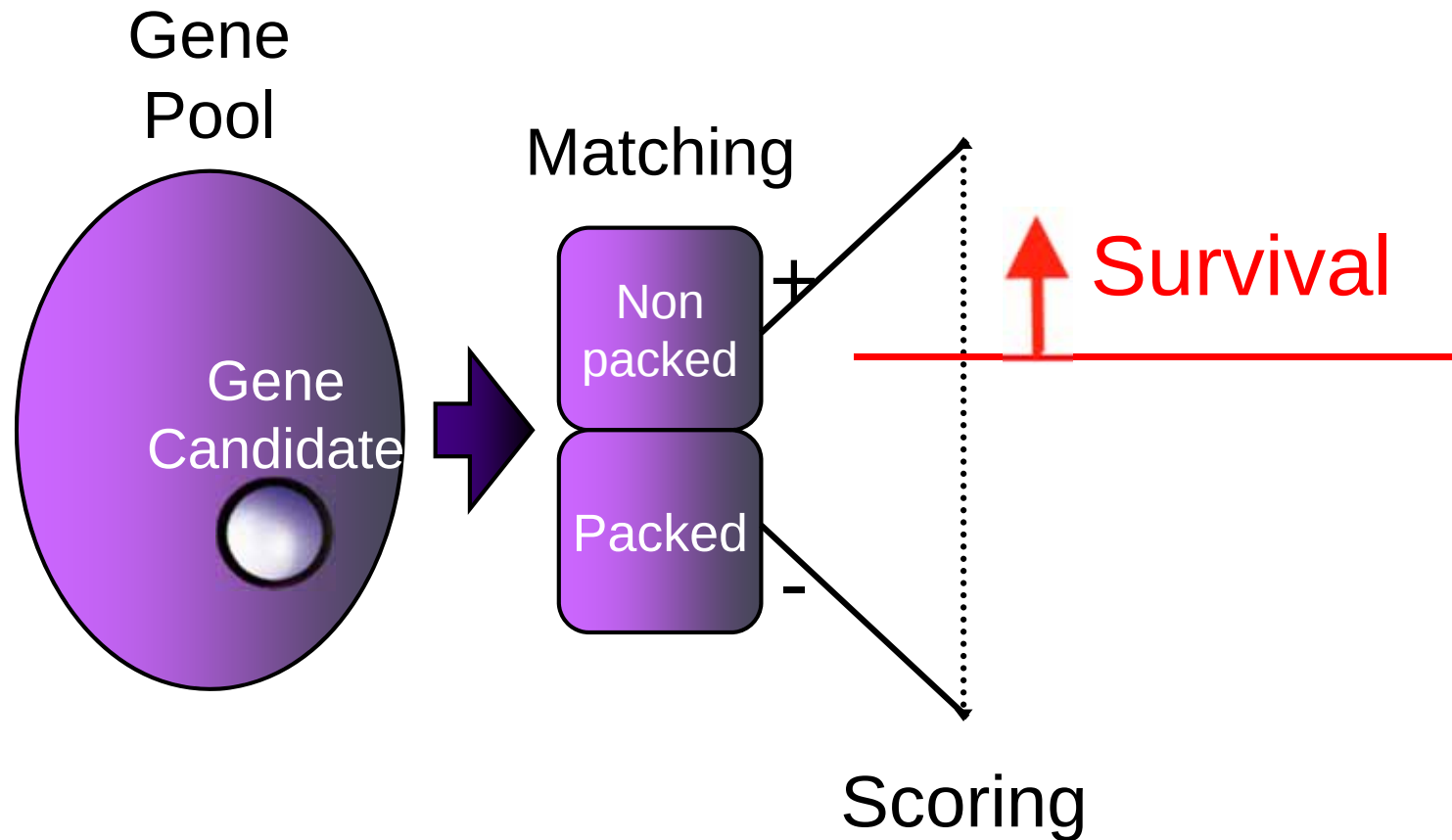


Genetics and packers



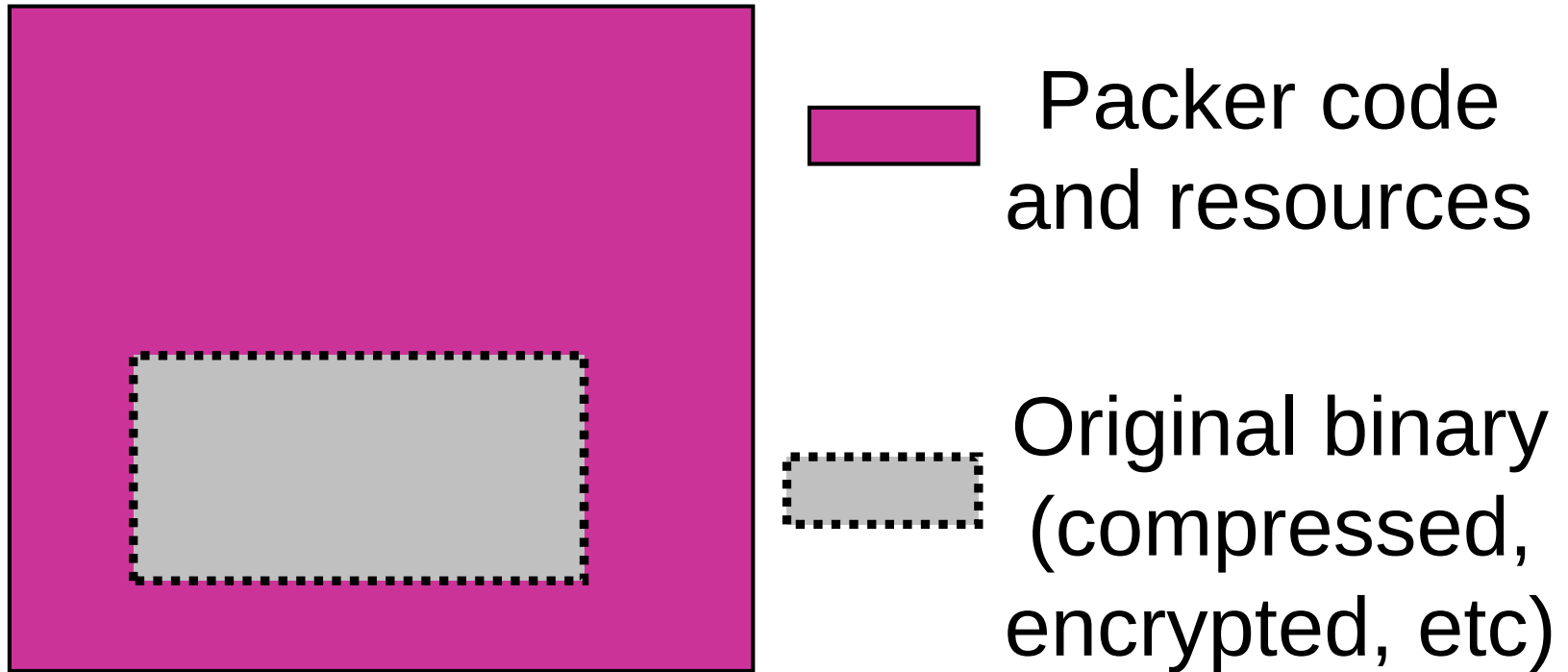
- Occurrences of pattern FF15 over a binary.
- 100% of hits inside code section.
- Good indicator of code density

Gene selection

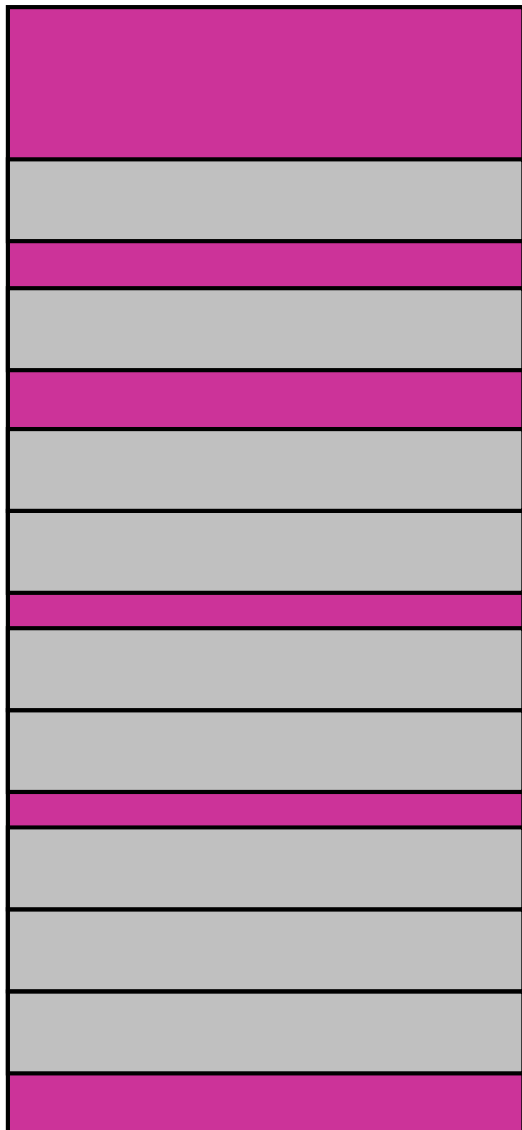


- Pythonized Bochs
- Compiler blood ties
- The selfish code
- Unpacking automation for
fun and profit

Packed executable



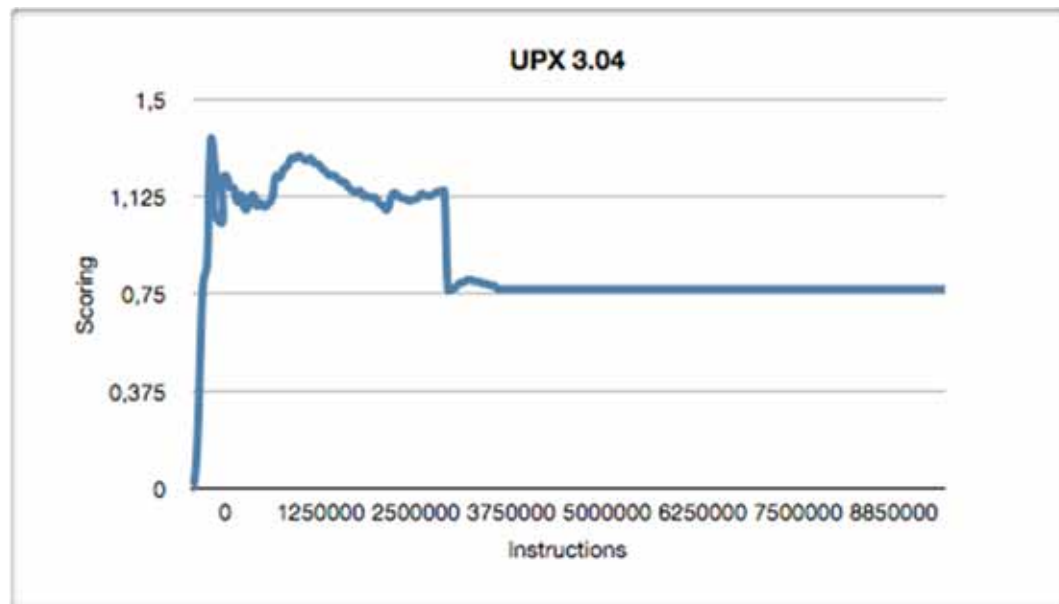
Unpacking process



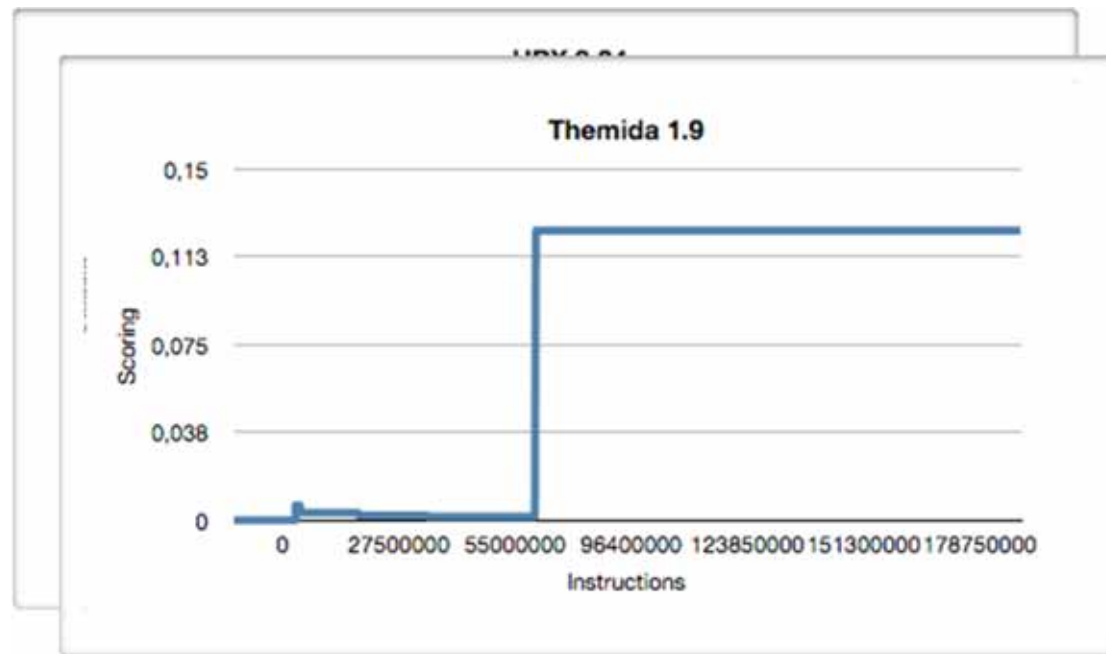
 Packer code
and resources

 Original code
(unpacked)

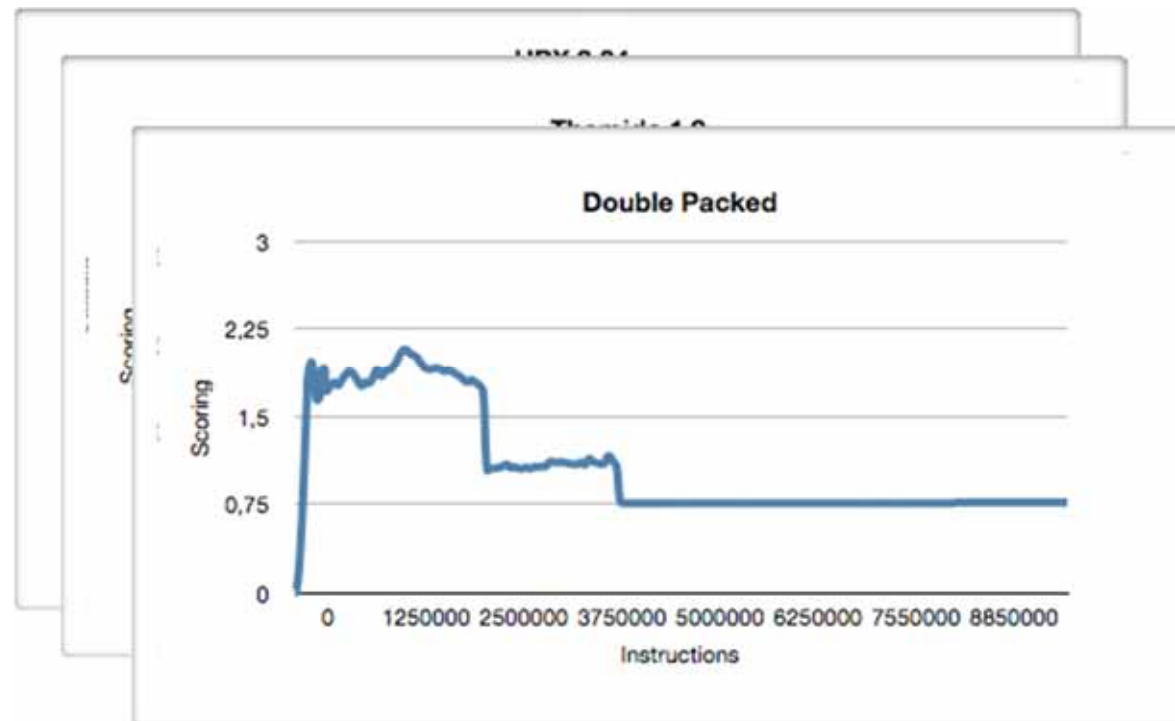
Gene[tr]ic unpacker



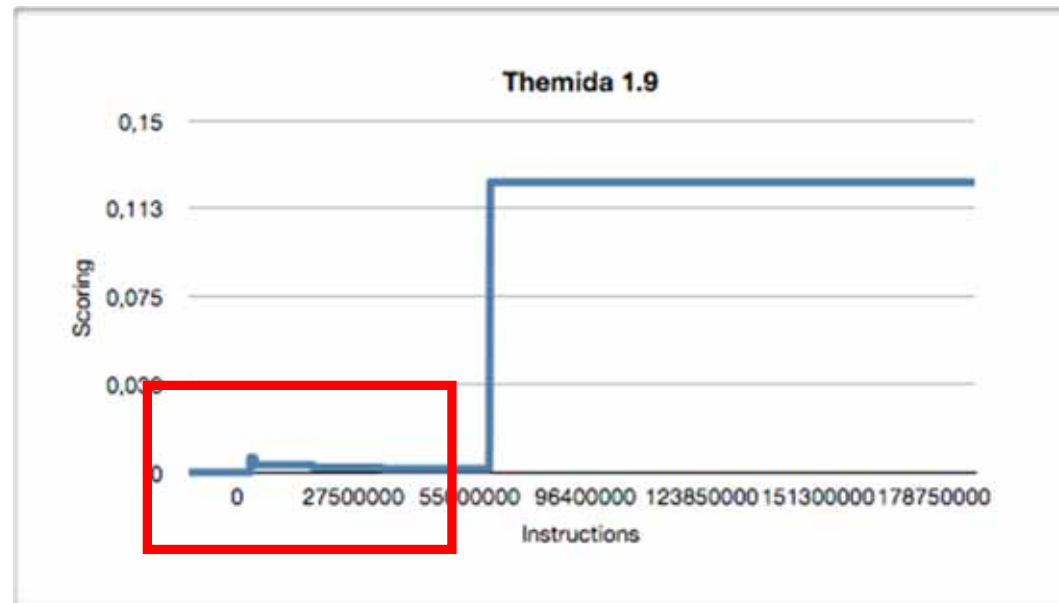
Gene[tr]ic unpacker



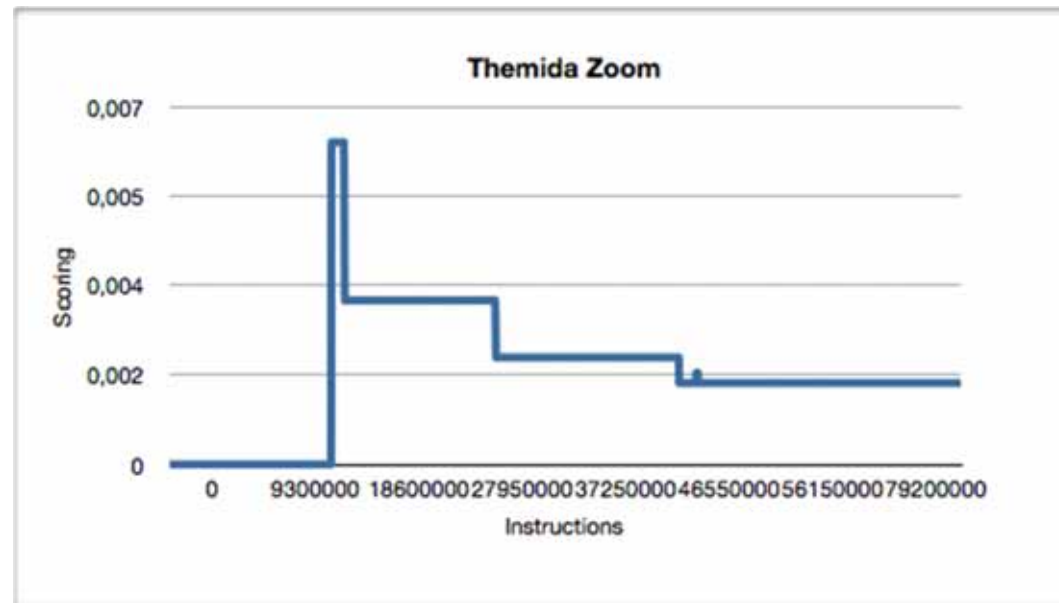
Gene[tr]ic unpacker



The stop problem



The stop problem



DEMO



Unpacking stats

- Test machine: VMware (1 core, 2Gb ram) running on a i7 2.8Ghz
- Minimum unpacking time: 5 seconds
- Maximum unpacking time: relative to MaxTicks (user configured)
- Average: 10.8 secs
 - Test performed over 580 packers (includes different versions and protection options)

Problems...

- ... that are not a problem:
 - Packers with compiler-code (Armadillo and Visual-C/WinMain)
 - N-packed samples

Problems...

- ... to solve:
 - Virtualization: the process works with really low scorings but we do nothing to analyze the VM
 - Obfuscation: similar issue with VMs
 - Not affected by anti-disasm tricks as we do pattern-matching
 - Decryption on demand: Code that changes after stabilization

Work in progress

- Gene[tr]ic unpacker as an online tool
- Plugins (optional post-processing):
 - Deobfuscator
 - VM analysis (as described in [1] to obtain more accurate callgraphs)
- Support for other CPUs, OSs and VMs

Questions?

